2020 © Adam Meissner

Politechnika Poznańska
Instytut Informatyki

Adam Meissner

Adam.Meissner@put.poznan.pl
http://www.man.poznan.pl/~ameis

SZTUCZNA INTELIGENCJA

Elementy wnioskowania automatycznego

Literatura

[1] Ben-Ari M., *Logika matematyczna w informatyce*, WNT, Warszawa, 2006.

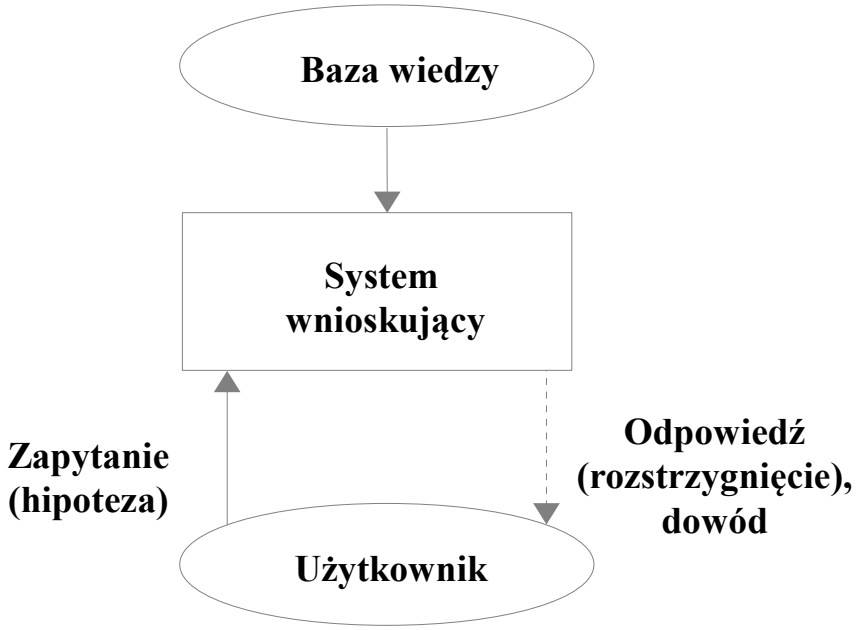
[2] Nilsson U., Małuszyński J., Logic, Programing and Prolog (2ed), John Wiley & Sons Ltd,1995.

[3] Stickel M., *A Prolog Technology Theorem Prover: A New Exposition and Implementation in Prolog*. SRI Int. Menlo Park, Technical Note 464, 1989.

[4] Wójcik M., *Zasada rezolucji*; *teoria*, *praktyka*, *kierunki rozwoju*; raport IPI PAN **662**, czerwiec 1989.

Wprowadzenie

- system wnioskowania automatycznego, inaczej: system wnioskujący (ang. *Reasoning System*) jest implementacją metody wnioskowania w danym rachunku formalnym, która obejmuję:
  - reguły wnioskowania
  - strategie wnioskowania
- architektura systemu wnioskującego



- przykładowe zastosowania systemów wnioskujących:
  - systemy ekspertowe
  - formalna weryfikacja oprogramowania
  - automatyczne dowodzenie twierdzeń
- do metod wnioskowania, najczęściej wykorzystywanych w systemach RS dla logiki I rzędu należy:
  - wnioskowanie rezolucyjne [Robinson, 1965]
  - dedukcja naturalna [Gentzen, 1934], w szczególności rachunek tabel analitycznych [Smullyan, 1968].

Reguła rezolucji

Def. (literał, literał pozytywny, literał negatywny)

Literałem nazywa się formułę atomową lub formułę atomową z negacją. Formuła atomowa jest *literałem pozytywnym*, a formuła atomowa z negacją jest *literałem negatywnym*. Dowolne literały postaci A oraz $\sim A$ stanowią parę *literałów komplementarnych*.

Def. (klauzula)

Klauzulą nazywa się dowolną formułę, przyjmującą jedną z wymienionych postaci:

1. $(\forall x_1) \ldots (\forall x_m) L_1 \vee \ldots \vee L_n$ dla $n > 1$,
2. $(\forall x_1) \ldots (\forall x_m) L_1$
3. Formuła pusta (tzw. *klauzula pusta*, oznaczana symbolem $\square$)

przy założeniu, że $L_1, \ldots, L_n$ dla $n > 1$ są dowolnymi literałami, nie zawierającymi żadnych zmiennych oprócz $x_1, \ldots, x_m$ dla $m \geq 1$. Przyjmuje się, że kwantyfikatory w zapisie klauzuli mogą być pominięte.

Def. (reguła rezolucji, rezolwenta, literał aktywny)

Niech $A \vee B_1 \vee \ldots \vee B_m$ oraz $\sim A \vee C_1 \vee \ldots \vee C_n$ będą dowolnymi klauzulami, gdzie A jest formułą atomową a $B_1, \ldots, B_m$ i $C_1, \ldots, C_n$ to dowolne literały. Regułę wnioskowania postaci

$$\frac{A \vee B_1 \vee \ldots \vee B_m, \sim A \vee C_1 \vee \ldots \vee C_n}{B_1 \vee \ldots \vee B_m \vee C_1 \vee \ldots \vee C_n}$$

nazywa się *regułą rezolucji*. Formuła $B_1 \vee \ldots \vee B_m \vee C_1 \vee \ldots \vee C_n$ nosi nazwę *rezolwenty* a literały $\sim A$ oraz A noszą nazwę *literałów aktywnych*.

Wnioskowanie rezolucyjne (1)

- wnioskowanie rezolucyjne jest metodą dowodzenia nie wprost
- wszystkie formuły przetwarzane w procesie wnioskowania mają postać klauzul
- dowolny zbiór *S* formuł logiki I rzędu można przekształcić w zbiór klauzul *S′* taki, że *S* jest niespełnialny wtw. *S′* jest niespełnialny.

Def. (rezolucja liniowa)

Rezolucja liniowa jest to strategia wnioskowania za pomocą reguły rezolucji, w której każdą, kolejną parę przesłanek tworzą następujące dwa elementy:

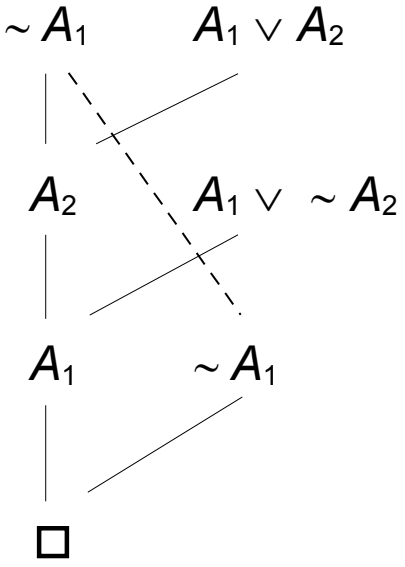
(1) w pierwszym kroku - negacja hipotezy, w każdym kolejnym kroku - rezolwenta uzyskana w kroku poprzednim,

(2) dowolny aksjomat, negacja hipotezy lub dowolna rezolwenta skonstruowana wcześniej.

Przykład 1

teoria : $\{A_1 \vee A_2\,,\, A_1 \vee \sim A_2\}$

hipoteza: A_1



Wnioskowanie rezolucyjne (2)

Def. (rezolucja źródłowa)

Rezolucja źródłowa jest to strategia wnioskowania za pomocą reguły rezolucji, w której każdą, kolejną parę przesłanek tworzą następujące dwa elementy:

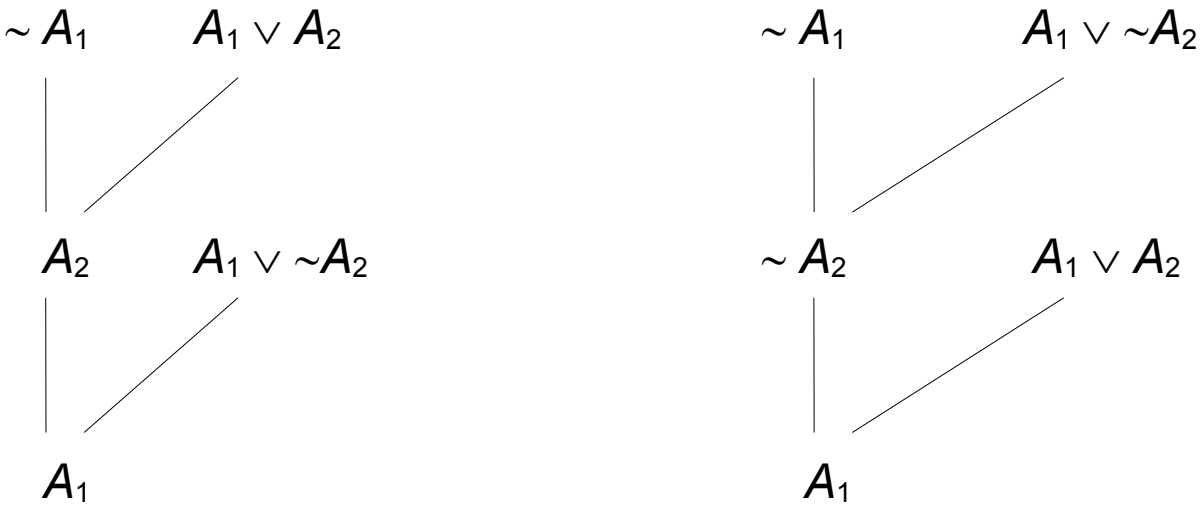
(1) w pierwszym kroku - negacja hipotezy, w każdym kolejnym kroku - rezolwenta uzyskana w kroku poprzednim,

(2) dowolny aksjomat.

Przykład 2 (niepełność rezolucji źródłowej)

teoria : $T = \{A_1 \vee A_2\,,\, A_1 \vee \sim A_2\}$

hipoteza: A_1



Hipotezy A_1 , dowodliwej w teorii T, nie można udowodnić za pomocą rezolucji źródłowej (tj. uzyskać klauzuli pustej) bez względu na sposób konstruowania wywodu (czyli kolejność wyboru przesłanek).

Wnioskowanie rezolucyjne (3)

Def. (uniwersum Herbranda)

Niech *S* będzie zbiorem klauzul i niech *Cnst* oraz *Fun* będzie odpowiednio zbiorem wszystkich stałych i zbiorem wszystkich symboli funkcyjnych, występujących w *S*. *Uniwersum Herbranda H* dla zbioru *S* określa się następująco:

$$H_0 = \begin{cases} Cnst, \text{ jeżeli } Cnst \neq \varnothing \\ \{a\}, \text{ jeżeli } Cnst = \varnothing \end{cases}$$

$H_{i+1} = H_i \cup \{f(t_1, .., t_n) \mid f \in Fun \wedge t_i \in H_i \text{ dla } i = 1, ..., n.$

$H = \cup H_i$ dla $i \in \mathbb{N}$

Przykład 3 (Uniwersum Herbranda)

Niech $S = \{ \sim p(x) \vee q(f(x))\}$. Wówczas:

$H_0 = \{a\}$, $H_1 = \{a, f(a)\}$, $H_2 = \{a, f(a), f(f(a))\}$, ...

$H = \{a, f(a), f(f(a)), f(f(f(a))), ...\}$

Def. (instancja podstawowa klauzuli)

Niech *S* będzie zbiorem klauzul zawierającym element *C*. *Instancją podstawową klauzuli C dla zbioru S* jest dowolna klauzula otrzymana z *C* poprzez zastąpienie wszystkich zmiennych występujących w *C* elementami uniwersum Herbranda dla zbioru *S*.

Twierdzenie Herbranda

Zbiór klauzul *S* jest niespełnialny wtedy i tylko wtedy gdy istnieje skończony, niespełnialny zbiór *S'* złożony z instancji podstawowych klauzul ze zbioru *S*.

Przekształcenie formuły w zbiór klauzul (1)

1. Wyeliminować z formuły spójniki $\leftrightarrow$ oraz $\leftarrow$ za pomocą reguł
   - $F \leftrightarrow G \equiv (F \leftarrow G) \wedge (G \leftarrow F)$
   - $F \leftarrow G \equiv F \vee \sim G$
2. Wprowadzić znak negacji bezpośrednio przed symbole atomowe za pomocą reguł
   - $\sim(\sim F) \equiv F$
   - $\sim(F \vee G) \equiv \sim F \wedge \sim G$
   - $\sim(F \wedge G) \equiv \sim F \vee \sim G$
   - $\sim(\forall\ x)\, F(x) \equiv (\exists\ x) \sim F(x)$
   - $\sim(\exists\ x)\, F(x) \equiv (\forall\ x) \sim F(x)$
3. Przesunąć kwantyfikatory maksymalnie na lewą stronę formuły za pomocą reguł
   - $(Q\,x)\, F(x) \vee G \equiv (Q\,x)\, (F(x) \vee G)$
   - $(Q\,x)\, F(x) \wedge G \equiv (Q\,x)\, (F(x) \wedge G)$
   - $(\forall\ x)\, F(x) \wedge (\forall\ x)\, G(x) \equiv (\forall\ x)\, (F(x) \wedge G(x))$
   - $(Q\ x)\, F(x) \vee (Q\ x)\, G(x) \equiv (Q\ x)\, (F(x) \vee G(x))$
   - $(Q\,x)\, F(x) \vee (Q'x)\, G(x) \equiv (Q\,x)(Q'y)\, (F(x) \vee G(y))$
   - $(Q\,x)\, F(x) \wedge (Q'x)\, G(x) \equiv (Q\,x)(Q'y)\, (F(x) \wedge G(y))$

Po przekształceniach 1 - 3, formuła wejściowa przybiera postać $(Q_1\ x_1) \ldots (Q_n\ x_n)\ M(x_1, \ldots, x_n)$. Część $(Q_1\ x_1) \ldots (Q_n\ x_n)$ nosi nazwę *przedrostka* formuły, a część $M(x_1, \ldots, x_n)$ nazywana jest *matrycą* formuły.

4. Wyprowadzić znak koniunkcji na zewnątrz wszystkich nawiasów za pomocą reguł
   - $F \vee\ (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$
   - $F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$

Przekształcenie formuły w zbiór klauzul (2)

5. Wyeliminować z przedrostka formuły kwantyfikatory szczegółowe za pomocą następujących reguł. Niech x_i dla $i = 1, ..., n$ będzie dowolną zmienną związaną przez kwantyfikator szczegółowy.

   Jeżeli kwantyfikator wiążący x_i nie jest poprzedzony w przedrostku przez żaden kwantyfikator ogólny, to usunąć $(\exists x_i)$ z przedrostka a wszystkie wystąpienia zmiennej x_i w matrycy $M(x_1, ..., x_n)$ zastąpić nową stałą s, która nie występuje wcześniej w matrycy.

   W przeciwnym razie, niech $\{y_1, ..., y_m\}$ będzie zbiorem wszystkich zmiennych związanych przez kwantyfikatory ogólne, które w przedrostku poprzedzają $(\exists x_i)$. Usunąć $(\exists x_i)$ z przedrostka, a wszystkie wystąpienia zmiennej x_i w matrycy zastąpić termem $f(y_1, ..., y_m)$, gdzie f jest nowym symbolem funkcyjnym niewystępującym wcześniej w matrycy.

   Proces realizowany w niniejszym kroku nosi nazwę *skolemizacj*i, a symbole s oraz f to odpowiednio *stała Skolema* i *funkcja Skolema*.

6. Po przekształceniach 1 - 5 formuła wejściowa przyjmuje postać $(Q_1\ x_1) ... (Q_p\ x_p)\ C_1 \wedge ... \wedge C_r$, gdzie formuły $C_1, ... , C_r$ są klauzulami. Utworzyć wyjściowy zbiór klauzul $\{C_1, ... , C_r\}$

Przykład 4

Przekształcić w zbiór klauzul formułę postaci:
$(\exists z)(\forall x)(r(z) \wedge p(x) \rightarrow (\exists y)(q(x,y) \wedge q(y)))$.

1. $(\exists z)(\forall x)(\sim(r(z) \wedge p(x)) \vee (\exists y)(q(x,y) \wedge q(y))$
2. $(\exists z)(\forall x)(\sim r(z) \vee \sim p(x) \vee (\exists y)(q(x,y) \wedge q(y))$
3. $(\exists z)(\forall x)(\exists y)(\sim r(z) \vee \sim p(x) \vee (q(x,y) \wedge q(y))$
4. $(\exists z)(\forall x)(\exists y)((\sim r(z) \vee \sim p(x) \vee q(x,y)) \wedge (\sim r(z) \vee \sim p(x) \vee q(y))$
5. $(\sim r(s) \vee \sim p(x) \vee q(x,f(x))) \wedge (\sim r(s) \vee \sim p(x) \vee q(f(x))$
6. $\{\sim r(s) \vee \sim p(x) \vee q(x,f(x)), \sim r(s) \vee \sim p(x) \vee q(f(x))\}$

Systemy wnioskowania automatycznego (RS)

- przykładem prostego systemu RS dla logiki pierwszego rzędu, w którym stosuje się wnioskowanie rezolucyjne jest PTTP (ang. *Prolog Technology Theorem Prover*) [3]
- system PTTP wykorzystuje środowisko wykonawcze języka Prolog jako efektywny mechanizm wnioskujący dla logiki klauzul Horna, uzupełniając je o następujące elementy:
  - *test wystąpień* (ang. *occur-check*), wprowadzany do algorytmu unifikacji
  - *eliminację modeli* (ang. *model elimintation*) jako nową strategię wnioskowania rezolucyjnego
  - strategię DFIDS jako metodę przeszukiwania drzewa wywodów

Unifikacja z testem wystąpień

- test wystąpień dla zmiennej *x* i termu złożonego *t* daje wynik pozytywny (*x występuje w t*), jeżeli *x* jest argumentem w *t* lub w dowolnym podtermie termu *t*
- jeżeli zmienna *x* występuje w termie *t*, to unifikacja *X* i *t* powinna zakończyć się niepowodzeniem, w przeciwnym wypadku wynikiem unifikacji jest term cykliczny, który nie należy do uniwersum Herbranda, np. *x* /*f*(*x*) daje wynik *f*(...*f*(...) ...)
- wiele środowisk wykonawczych języka Prolog w trakcie unifikacji nie przeprowadza testu wystąpień z uwagi na jego złożoność obliczeniową oraz fakt, że termy cykliczne, mimo iż "formalnie niepoprawne", dobrze nadają się do reprezentowania niektórych struktur danych (np. grafów cyklicznych)
- w systemie PTTP test wystąpień wykonuje się za pomocą predykatu `unify_with_occurs_check/2` podczas unifikowania bieżącego celu z nagłówkiem klauzuli należącej do programu reprezentującego wejściowy zbiór formuł

Przykład 5

Niech będzie dany aksjomat *A*, który intuicyjnie może np. oznaczać, że dla każdej liczby istnieje liczba większa od niej:

$$(\forall x)(\exists y)\ greater(y, x)$$

. Po skolemizacji, formuła *A* przyjmuje postać:

$$greater(f(x), x)$$

Stosując wnioskowanie rezolucyjne bez testu wystąpień można udowodnić na podstawie *A* formułę nieprawdziwą, stwierdzającą że istnieje liczba większa od samej siebie, tj. $(\exists x)\ greater(x, x)$:

$$\frac{\sim greater(x,x),\ greater(f(x),\ x)}{\square,\ = \{x/f(x)\}}$$

Eliminacja modeli (1)

- SLD-rezolucja, wykorzystywana do wnioskowania przez środowiska wykonawcze języka Prolog, jest szczególnym przypadkiem rezolucji źródłowej, która nie jest strategią pełną dla klauzul dowolnych
- SLD-rezolucję można łatwo rozszerzyć do strategii eliminacji modeli, która ma własność pełności w logice pierwszego rzędu
- strategia eliminacji modeli działa na wyrażeniach nazywanych *łańcuchami* – są to ciągi składające się w ogólności z literałów oraz tzw. A-literałów, zapisywanych poniżej w nawiasach kwadratowych; dowolnej klauzuli $L_1 \vee ... \vee L_n$ odpowiada łańcuch $L_1, ..., L_n$
- eliminacja modeli obejmuje 3 reguły wnioskowania; niech H i H' będą atomami a symbole Q oraz L (z indeksami dolnymi) niech oznaczają literały:
  - rozszerzanie łańcucha (*ro*) - niech $H\Theta = H'\Theta$, wówczas

$$\frac{\sim H,\ Q_1,\ \ldots\ ,\ Q_m,\ \ H' \vee L_1 \vee \ldots \vee L_n}{(\ L_1,\ldots,\ L_n,\ [\sim H],\ Q_1,\ \ldots,\ Q_m)\Theta}$$

  - redukcja łańcucha (*re*) - niech $H\Theta = Q_i\Theta$, wówczas

$$\frac{\sim H,\ Q_1,\ \ldots,\ [Q_i],\ \ldots,\ Q_m}{Q_1,\ \ldots,\ [Q_i],\ \ldots,\ Q_m}$$

  - usuwanie A-literału (*u*)

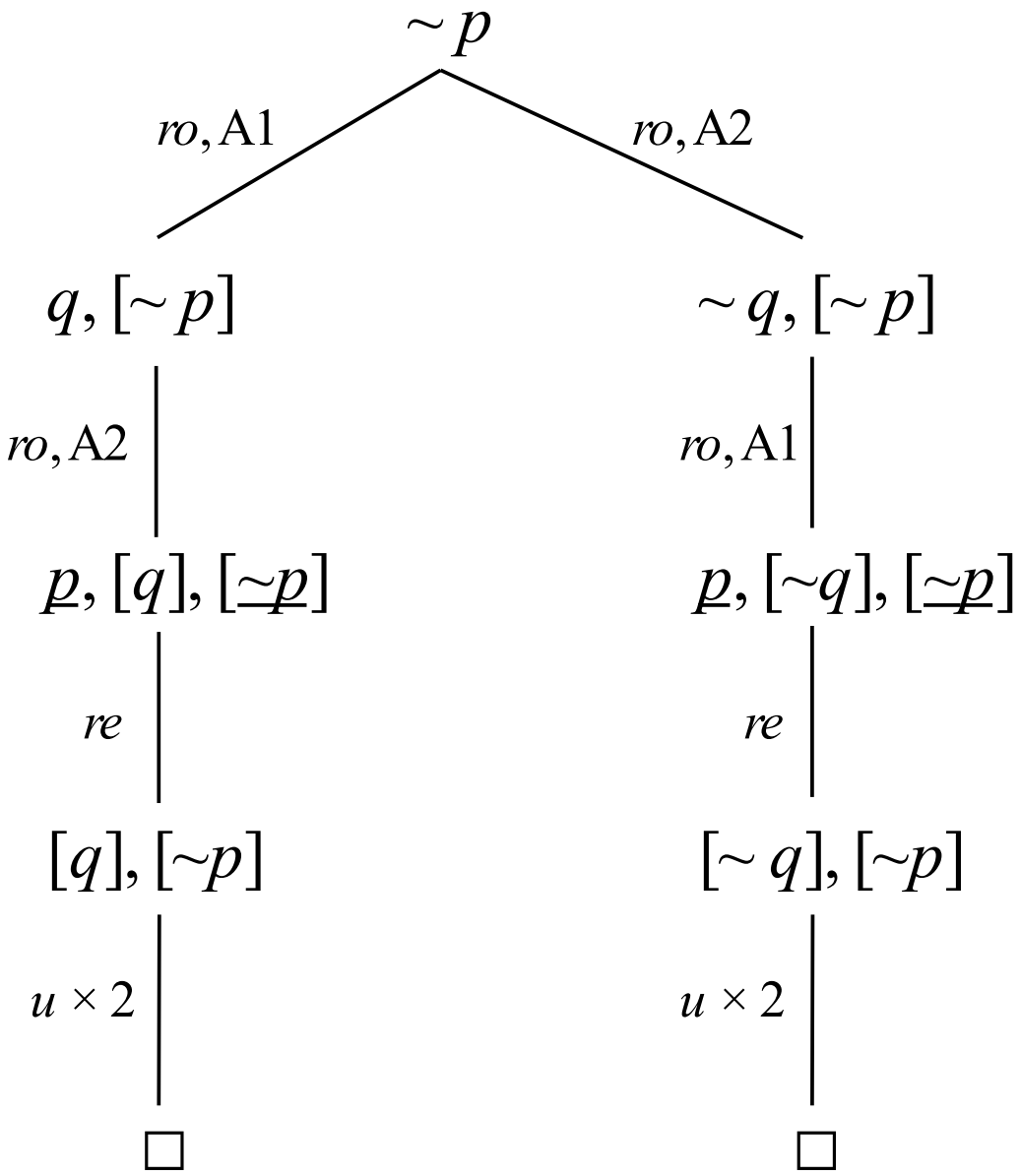
$$\frac{[Q_1],\ Q_2,\ldots,\ Q_m}{Q_2,\ \ldots,\ Q_m}$$

- eliminację modeli w systemie PTTP implementuje się poprzez konstruowanie listy *Anc*, zawierającej kolejne cele, których "wywoływanie" doprowadziło do celu bieżącego - jeżeli cel bieżący unifikuje się z negacją dowolnego elementu listy *Anc*, to należy cel ten pominąć w rezolwencie

Eliminacja modeli (2)

Przykład 6

Dane jest zapytanie (hipoteza) p, kierowane do bazy wiedzy zawierającej następujące dwie formuły (aksjomaty): $p \vee q$ (A1) oraz $p \vee \sim q$ (A2). Dowód hipotezy p, za pomocą strategii eliminacji modeli, można przeprowadzić na dwa sposoby przedstawione poniżej.



Strategia DFIDS

- środowiska wykonawcze języka Prolog przeszukują drzewa wywodów za pomocą strategii DFS (ang. *Depth-First Search*); nie jest to strategia pełna w tym sensie, że dowód rozpatrywanej hipotezy nie zostanie znaleziony, o ile w drzewie "poprzedza" go wywód nieskończonej długości
- system PTTP używa do przeszukiwania drzewa wywodów strategii DFIDS (ang. *Depth-First Iterative-Deepening Search*), czyli przeszukiwania w głąb z iteracyjnym pogłębianiem, która w skończonej liczbie kroków może znaleźć dowolną skończoną ścieżkę w drzewie a ponadto jest optymalna w zbiorze strategi przeszukiwania siłowego z uwagi na długość pierwszego znalezionego dowodu, czas jego znalezienia oraz wykorzystanie pamięci
- w implementacji strategii DFIDS wykorzystuje się predykat `gen_depth(+B,+S,-D)`, który w toku kolejnych nawrotów nadaje zmiennej `D` wartości `B, B+S, B+2*S, B+3*S, ...`

Przykład 7

Dany jest program *P* w języku Prolog:

```
r(X) :- r(f(X)).
r(0).
```

Środowisko wykonawcze Prologu, wykorzystujące strategię DFS, nie znajdzie dowodu hipotezy *Q* postaci `:- r(0).` na podstawie programu *P* pomimo, że $P \models Q$.

System PTTP

Dane wejściowe

Na dane wejściowe systemu składa się reprezentacja zbioru aksjomatów *T* oraz hipotezy *Q*. Reprezentację zbioru aksjomatów *T* konstruuje się następująco.

1. Przekształcić zbiór aksjomatów *T* w zbiór klauzul *K*.
2. Dla każdego elementu zbioru *K* postaci $L_1 \vee ... \vee L_i \vee ... \vee L_n$ skonstruować zbiór klauzul prologowych *P* postaci

L'_1 :- $\sim L'_2$, ..., $\sim L'_n$.
...
L'_i :- $\sim L'_1$, ..., $\sim L'_{i\text{-}1}$, $\sim L'_{i+1}$, ..., $\sim L'_n$.
...
L'_n :- $\sim L'_1$, ..., $\sim L'_{n\text{-}1}$.

Suma wszystkich ww. zbiorów jest reprezentacją zbioru aksjomatów *T*. Symbol L' oznacza prologową reprezentację literału L. Ze względu na zastąpienie negacji przez porażkę metodą eliminacją modeli, każdy literał negatywny występujący w powyższym zbiorze reprezentuje się za pomocą literału pozytywnego, którego nazwa rozpoczyna się od wyróżnionego przedrostka *not_*. Przykładowo, literał $\sim p(x)$ jest reprezentowany jako `not_p(X)`.

Reprezentacją hipotezy *Q* jest formuła *Q'* postaci $D_1 ; ... ; D_m$ gdzie wyrażenie D_i dla $i = 1, m$ jest koniunkcją literałów. Do skonstruowania formuły *Q'* wykorzystuje się metodę podobną do przekształcania dowolnej formuły w zbiór klauzul, z uwzględnieniem następujących różnic.

1. W kroku 4 na zewnątrz wszystkich nawiasów wyprowadza się znak alternatywy.
2. Przed wykonaniem kroku 5 (skolemizacja) "odwraca się" kwantyfikatory.
3. Pomija się krok 6.

Przykład 8

Dany jest zbiór *T* = $\{(\forall x)(p(x) \vee (q(x) \wedge r(x))), (\forall x)(\sim p(x) \vee (q(x) \wedge r(x)))\}$ oraz hipoteza *Q* = $(\forall x)(q(x) \wedge r(x))$. Prologowa reprezentacja zbioru aksjomatów *T* ma postać następujących klauzul.

```
p(X) :- not_q(X).        r(X) :- not_p(X).
p(X) :- r(X).            r(X) :- p(X).

q(X) :- not_p(X).        not_p(X) :- not_q(X).
q(X) :- p(X).            not_p(X) :- not_r(X).
```

Reprezentacją hipotezy *Q* jest formuła *Q'* = `q(s),r(s)` gdzie `s` jest stałą Skolema.

Metainterpreter Prologu

- termin *metainterpreter* oznacza interpreter zdefiniowany w języku, który jest jednocześnie językiem interpretowanym
- szczególnym przykładem metainterpretera jest tzw. metainterpreter kołowy (ang. *metacircular interpreter*) zawierający środki wystarczające do przetwarzania samego siebie
- poniższy program to tzw. metainterpreter waniliowy (ang. *vanilla meta-interpreter*) – przymiotnik *vanilla* w przenośni oznacza tyle co zwykły, typowy, standardowy
- program w Prologu jest zbiorem klauzul Horna zapisywanych w postaci implikacyjnej wśród których wyróżnia się reguły `A :-` A_1`,...,`A_n`.` , gdzie `A` (z ew. indeksem dolnym) jest formułą atomową oraz fakty `A :- true.`
- metainterpreter ma postać definicji predykatu `prove/1`, którego argumentem jest zapytanie kierowane do interpretowanego programu; program ten jest umieszczony (wraz z metainterpreterem) w prologowej bazie danych

```
prove((A,B)) :-                          % 1
   prove(A),                             % 2
   prove(B).                             % 3

prove((A;B)) :-                          % 4
   prove(A);                             % 5
   prove(B).                             % 6

prove(true).                             % 7

prove(Query) :-                          % 8
   clause(Query,Body),                   % 9
   prove(Body).                          % 10
```

Implementacja systemu PTTP

System PTTP ma postać przedstawionego niżej programu w Prologu. Po wprowadzeniu do prologowej bazy danych reprezentacji zbioru aksjomatów *T* program można uruchomić przez sformułowanie celu `:-prove(Query)`, gdzie wartością zmiennej `Query` jest formuła *Q*'. Formuła `compl_mem(H,Anc)` w linii 13 jest prawdziwa o ile lista `Anc` zawiera literał komplementarny do literału *H* w sensie negacji reprezentowanej przez przedrostek `not_`. Przykładowo, prawdziwa jest formuła `compl_mem(not_p(a),[p(X)])`. Uzyskana odpowiedź jest twierdząca wtedy i tylko wtedy gdy $T' \models Q'$.

```
prove(Query) :-
   gen_depth(Depth),                          % 1
   prove(Query,[],Depth).                     % 2

prove(_,_,Depth) :-                           % 3
   Depth < 0,!,fail.                          % 4

prove((A,B),Anc,Depth) :- !,                  % 5
   prove(A,Anc,Depth),                        % 6
   prove(B,Anc,Depth).                        % 7

prove((A;B),Anc,Depth) :- !,                  % 8
   prove(A,Anc,Depth);                        % 9
   prove(B,Anc,Depth).                        % 10

prove(true,_,_) :-!.                          % 11

prove(H,Anc,_) :-                             % 12
   compl_mem(H,Anc).                          % 13

prove(H,Anc,Depth) :-                         % 14
   functor(H,Name,Arity),                     % 15
   functor(NewH,Name,Arity),                  % 16
   clause(NewH,B),                            % 17
   unify_with_occurs_check(NewH,H),           % 18
   Depth1 is Depth - 1,                       % 19
   prove(B,[H|Anc],Depth1).                   % 20
```